View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Theseus

Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Yrityksen tietoliikenne ja tietoturva

2015

Ville Kukkonen

ENSIMMÄINEN PELI UNITY-PELIMOOTTORILLA

– kenttäsuunnittelun implementointi

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Yrityksen tietoliikenne ja tietoturva

Joulukuu 2015 | 35

Jarkko Paavola

Ville Kukkonen

ENSIMMÄINEN PELI UNITY-PELIMOOTTORILLA – KENTTÄSUUNNITTELUN IMPLEMENTOINTI

Opinnäytetyön tavoitteena oli selvittää, onko mahdollista luoda peli tai pelidemo Unityn pelimoottoria hyödyntäen ilman aikaisempaa kokemusta. Toteutettu pelidemo oli kaksiulotteinen kauhupeli, joka on luotu teoriaosuudessa käytyjen pohdintojen mukaisesti. Opinnäytetyö käsittelee pelin luomista kenttäsuunnittelun näkökulmasta ja vastaa sen pohjalta asetettuihin kysymyksiin.

Teoriaosuudessa tarkastellaan kenttäsuunnittelua ja miten sitä kannattaa toteuttaa. Osuudessa käydään myös läpi kuvankäsittelyä sekä kuvankäsittelyohjelmia. Teorian tarkoituksena on antaa lukijalle karkea yleiskäsitys kenttäsuunnittelusta.

Toisessa osiossa pohjustetaan Unityn pelimoottoria, jolla pelidemo on toteutettu. Tuetut ohjelmointikielet, Unityn verkkokauppa sekä opetusmateriaalia selostetaan osiossa. Toisen osion tarkoituksena on kertoa tärkeää taustatietoa Unityn pelimoottorista.

Kolmannessa osiossa perehdytään pelimoottorin toimintoihin ja miten kenttäsuunnittelu toteutetaan käytännössä. Osio keskittyy ohjelmakoodin esittämiseen ja toimintojen selostukseen. Käytetyt toiminnot ovat pelidemosta poimittuja.

Opinnäytetyön lopputuloksena saatiin aikaiseksi onnistunut pelidemo ja samalla vastaus opinnäytetyön asettamaan tavoitteeseen. Peli on mahdollista toteuttaa ilman aikaisempaa kokemusta, mutta se vie paljon aikaa sekä resursseja.

ASIASANAT:

Peliohjelmointi, pelisuunnittelu, kuvankäsittely, Unity, pelimoottori

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Business Information Communications and Information Security

December 2015 | 35

Jarkko Paavola

Ville Kukkonen

FIRST GAME WITH UNITY GAME ENGINE – IMPLEMENTATION OF LEVEL DESIGN

The objective of this thesis was to find out if it is possible to create a game or a game demo using Unity's game engine without prior knowledge. The created game demo is a two dimensional horror game that is created by applying the theory presented in this thesis. The thesis covers game level design in theory and in practice.

The thesis examines game level design and how one should implement it at theoretical level. In this theoretical part, image processing and programs related to it are also explained. The purpose of this part was to give the reader a basic knowledge on game level design.

The second part of this thesis introduces the Unity game engine and its basic features and functions. It examines different programming languages, online store and learning material that are related to Unity. This part provides the must know facts about the Unity game engine.

The final part of the thesis explains the actual game engine and what features it contains. All the coding scripts and different features that are in the game demo are shown and explained thoroughly.

As the result of this thesis a successful game demo was created and the main objective of this thesis was achieved. It is possible to create a game with Unity without prior knowledge of it, but it is quite time consuming and uses several resources to do so.

KEYWORDS:

Game programming, game design, image processing, Unity, game engine

SISÄLTÖ

LIITTEET

KUVAT

TAULUKOT

SANASTO

2D	Kaksiulotteinen
3D	Kolmiulotteinen
Asset	Valmispaketti. Asset on valmis grafiikan tai pelimekaniikan osa, jota voi käyttää suoraan Unityn pelimoottorissa. Valmispaketteja voi ostaa tai ladata Unityn verkkokaupasta
Avatar	Pelaajan liikuttama pelihahmo
Editori	Muokkausohjelma, jolla muutetaan tai luodaan asioita. Tässä kontekstissa Unity on editori, jolla luodaan pelejä
Kenttä	Monesta tasosta koostuva yhtenäinen osio. Tasot ovat hyvin samankaltaisia yhdessä kentässä
Maailma	Pelin alue, mikä pitää sisällään eri kenttiä. Pelimaailmassa on tiettyjä samoja elementtejä, ajatuksia ja konsepteja, mutta ne voivat vaihdella suurestikin
Pelimoottori	Valmis rakenne, jota hyödyntämällä ei tarvitse kehittää kaikkia pelinosasia itse. Sisältää fysiikkamallinnuksia ja luontielementtejä
Peliobjekti	Unityssa oleva muokattava asia, toiminto, esine tai tapahtuma, jossa voi olla kuvia, koodia, fysiikkaelementtejä tai muita elementtejä
Taso	Yksi kentän osa, jossa avatar liikkuu paikasta A paikkaan B
Tutoriaali	Valmis opastus jonkin tietyn asian luontiin (yleensä videomuodossa). Esimerkiksi opastus miten tehdään pelihahmo ja saadaan se liikkumaan

1 JOHDANTO

Tässä opinnäytetyössä tutkitaan kenttäsuunnittelun merkitystä ja sen toteuttamista Unityn pelimoottorilla. Aiheesta on paljon materiaalia ja kursseja, mutta ne ovat usein hyvin henkilökohtaisia sekä omaan kokemukseen perustuvia havaintoja ja oppeja. Siksi opinnäytetyö tarkastelee kuinka on mahdollista toteuttaa videopeli ilmaisia resursseja hyödyntäen ja ilman aikaisempaa kokemusta.

Opinnäytetyön tarkoituksena on vastata kysymykseen, onko mahdollista toteuttaa peli Unitylla ilman aikaisempaa kokemusta. Työtä on tehty konstruktiivista tutkimusotetta käyttäen. Ratkottu ongelma on pelin luomista Unityn pelimoottorilla ilmaisia materiaaleja hyödyntäen. Opinnäytetyön aikana käydään läpi teoriaa sekä käytännön esimerkkejä, joilla ongelmaan saadaan vastauksia.

Peliprojektin tavoitteena oli luoda kaksiulotteisen kauhupelin demo Unityn pelimoottoria käyttäen. Projektin jäseniin kuului kolme henkilöä, joilla jokaisella oli oma roolinsa. Tarkoituksena oli saada pelidemo luotua omilla taidoilla ja resursseilla ilman valmiita maksullisia työkaluja ja materiaaleja. Peli luotiin Windows-käyttöjärjestelmille toimivaksi. Pelimoottoriksi valittiin Unity, koska se on ilmainen ja tarjoaa laajan valikoiman toimintoja, joita pelinkehitykseen tarvitaan. Unityn hyvä maine sekä reilu käyttöpolitiikka olivat iso tekijä pelimoottorin valitsemisessa. Pelimoottorista on myös saatavilla paljon opetusmateriaalia.

Tämä opinnäytetyö keskittyy pelikentän suunnittelemiseen aina ideatasosta käytännön tekemiseen pelimoottorissa. Ensiksi perehdytään kenttäsuunnittelun teoriaan. Työssä tarkastellaan, mitä ideoinnissa sekä luonnostelussa kannattaa ottaa huomioon. Teeman tärkeyttä tuodaan esille, sillä se on olennaisesti sidoksissa kentän visuaaliseen ilmeeseen. Koska teeman yhteys ideointiin ja kentän ulkonäköön on tärkeää, ei yhtenäisyyttä saa unohtaa. Projektin pienuudesta johtuen visuaalinen ilme määriteltiinkin yhdessä, mutta isoissa pelistudioissa sen hoitaa oma yksikkönsä. Projekti tehtiin ilman rahallista panostusta, joten teoriaosuudessa käydään läpi ilmaisia kuvankäsittelytyökaluja. Osio käsittelee sitä mitkä ovat hyviä ohjelmia ja mitä itse suosittelen.

Kuvankäsittelyohjelmien jälkeen pohjustetaan itse Unityn pelimoottoria. On tärkeää tietää Unitysta taustatietoa. Esimerkiksi mitä ohjelmointikieliä se tukee tai kuinka hyvin oppimateriaalia on saatavilla. Tässä osiossa tarkastellaan mitä kannattaa tietää ennen Unityn käyttöä.

Ohjelmointi on suuressa osassa pelikentän luonnissa, jonka vuoksi tuettujen ohjelmointikielien – C#, JavaScript sekä Boo – tärkeyttä on selitetty opinnäytetyössä. Osion lopussa tarkastellaan Unityn omaa verkkokauppaa, Asset Storea, josta voi ostaa tai ladata ilmaiseksi valmista grafiikkaa, toimintoja tai lisäosia.

Pelimoottorin pohjustuksen jälkeen tarkastellaan itse pelikentän luomista Unityyn. Osiossa kerrotaan projektin aloittamisesta sekä Unityn käytöstä. Projektin pelikentän tärkeimmät toiminnot ja pelimekaniikka on jaoteltu omiin kappaleisiin, joissa käydään yksityiskohtaisesti läpi ohjelmointikomentoja sekä niiden käyttötarkoitusta.

Yhteenvedossa pohditaan, kuinka hyvin projekti vastasi alkuperäiseen kysymykseen. Mitkä asiat tulivat projektissa yllätyksenä, mikä oli helppoa ja mitä opittiin. Lopuksi pohditaan myös, miten työtä voidaan kehittää jatkossa ja mitä parannettavaa projektissa on. Yhteenvedossa pohditaan esimerkiksi kuinka paljon työn määrä poikkesi alkuperäisestä suunnitelmasta, mitä realiteetteja huomattiin projektin edetessä sekä kuinka paljon alkutavoitteet vaihtelivat lopputuloksesta.

2 KENTTÄSUUNNITTELUN ESIVAIHEET

Ennen kenttäsuunnittelun aloittamista on erityisen tärkeää, että koko projektitiimin kanssa on suunniteltu pelin yhteinen teema, juoni sekä tapahtumapaikka ja -aika. Nämä kaikki vaikuttavat oleellisesti tulevien kenttien sekä tasojen ja koko pelimaailman luomiseen. Jos on päätetty pelin sijoittuvan 40-luvun natsi-Saksaan, ei kenttäsuunnittelija voi luoda kenttää tai tasoa, jossa ollaan amerikkalaistyylisessä hampurilaisravintolassa. Tämä rikkoisi pelin teeman ja tuntuisi irralliselta muihin kenttiin verrattuna.

Pelin teema on suurin tekijä kenttäsuunnittelua tehdessä. Se määrittelee yleisilmeen sekä tunnelman ja vaikuttaa taustalla koko ajan. Teemasta poikkeaminen rikkoo pelaajan pelikokemusta ja pahimmassa tapauksessa saa pelaajan lopettamaan pelaamisen kokonaan (Rouse III 2001, 9-10). Pelidemon teemana oli Keski-Suomen mökkimaisema ja toisen maailmansodan aikainen bunkkeri, joka löytyy mökkialueelta. Tällöin pitää kenttäsuunnittelussa ottaa huomioon, millaista maisemaa Keski-Suomessa on ja miltä toisen maailmansodan aikaiset bunkkerit ovat näyttäneet ja millaista kalustoa ne ovat sisältäneet.

Peliprojektin pienuuden sekä budjetin puuttumisen vuoksi on syytä muistaa, että kaikki osallistuvat ideointiin ja kenttäsuunnittelijan roolin omaava pyrkii toteuttamaan yhdessä suunnitellut ideat ja ajatukset. Jatkuva kommunikointi on tärkeää, sillä pienessä projektissa jokaisella on omat ajatuksensa pelistä ja olisi hyvin tärkeää, että kaikkien tuomat ideat saadaan julki. Näin saadaan luotua peli, johon kaikki projektin jäsenet ovat tyytyväisiä ja tällä varmistetaan, että pelistä tulee mahdollisimman yhtenäinen.

2.1 Luonnostelu

Ennen kenttien tekemistä pelimoottoriin pitää olla jokin idea pelikentästä sekä maailmasta, mihin peli sijoittuu. Aluksi eri pelikentät voivat sisältää monta eri ideaa ja ajatusta, joista karsitaan myöhemmin ylimääräiset pois. On myös tärkeää muistaa, ettei ole olemassa huonoja ideoita vaan kaikki tulisi kirjoittaa muistiin paperille (Rogers 2010, 36). Kun ideoita alkaa olla tarpeeksi, voidaan aloittaa kenttien muodostaminen ja niiden hahmottaminen. Tässä vaiheessa ei vielä tarvitse olla valmiiksi piirrettyjä kenttiä, pelkkä ideataso riittää.

Kun alkaa luonnostella varsinaista kenttää, kannattaa keskittyä yhteen ideaan tai ajatukseen, jolla vie kentän kehitystä eteenpäin. Idea voi olla joko pelimekaniikkaan sidottu tai visuaalinen idea (Bates 2004, 108). Tärkeää on kuitenkin, että kentällä on johdonmukainen suunta, joka kantaa luotua ideaa eteenpäin.

Tämä ei kuitenkaan tarkoita, että kentän pitäisi toimia mekaanisesti yhden toiminnon ympärillä. Vaihtelevuus on tärkeää myös yhden kentän sisällä, kunhan pääidea pysyy muuttumattomana. Eri kentät voivat puolestaan sisältää paljonkin eri ideoita. Olisi tärkeää kuitenkin säilyttää koko pelin läpi yhteinen teema, jotta eri kentät eivät tuntuisi irrallisilta. Siksi onkin tärkeää luonnostella ja hahmotella ideoita paperille ennen kuin aloitetaan kenttien luominen pelimoottorissa (Bates 2004, 109.)

2.2 Uskottavuuden saavuttaminen

Pelin pitää olla aidon tuntuinen. Tätä ei saada aikaan suoraan graafisella ulkoasulla vaan asioiden esittämisellä siten, että ne voisivat tapahtua oikeassa elämässä. Esineiden, asioiden ja ihmisten pitää näyttää, toimia ja käyttäytyä samalla tavalla kuin meidän maailmassamme. Pelaajan pitää samaistua pelimaailmaan. Samaistuminen tapahtuu asioilla, jotka tuntuvat uskottavilta. Tämä ei tarkoita, että fantasiapelissä ei saa olla lohikäärmeitä, koska niitä ei oikeasti ole olemassa. Lohikäärme pitää esittää maailmassa uskottavasti esimerkiksi siten, että se käyttäytyy kuin kansantaruissa on esitetty ja kuvailtu.

Uskottavuus syntyy pelin sisäisistä asioista, jotka mallintavat oikeaa elämää. Pelaajan tulisi ymmärtää pelimaailman sisällä, mitkä teot toimisivat maailman sääntöjen mukaan (Rouse III 2001, 9). Tavallisten esineiden ja asioiden tulisi käyttäytyä pelaajan olettamalla tavalla. Esimerkiksi, jos pelaaja menee pelissä kauppaan, hän olettaa saavansa ostettua tavaraa puhumalla myyjälle. Myyjä ei voi yhtäkkiä muuttua liskoksi ja hyökätä pelaajan kimppuun, kun pelaaja on tullut kauppaan vain ostamaan hahmolleen tavaroita.

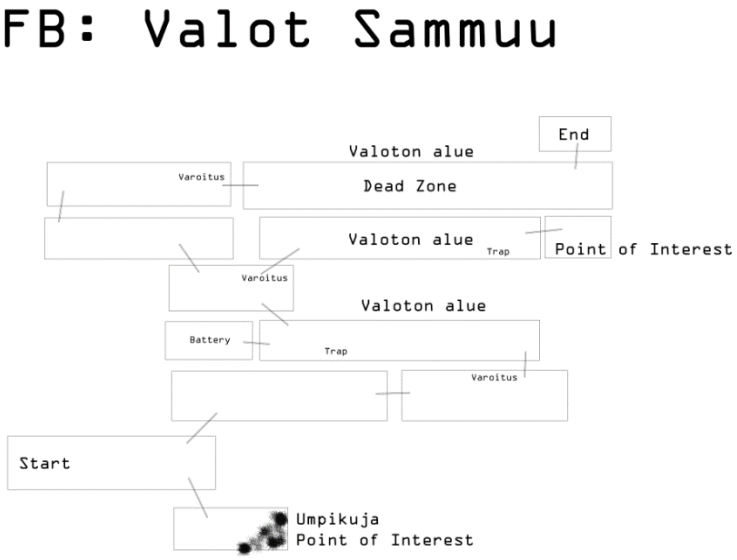
Pelimaailman asettamat rajat ja säännöt on tärkeää näyttää pelaajalle jo alkuvaiheessa, jotta pelaaja osaa olettaa tiettyjä asioita peliltä. Kun pelaaja odottaa tiettyä loogista lopputulosta pelissä, mutta ilman mitään hyvää syytä tapahtuu jotain muuta, on tämä pelaajalle hyvin turhauttavaa. Vielä pahempaa on kun pelaajan tekemät toiminnat tuottavat täysin arvaamattoman tapahtuman. Lopputuloksena pelaaja turhautuu ja todennäköisesti etsii toisen järjestelmällisemmän pelin (Rouse III 2001, 9.)

Kun peli toimii oikean elämän tavoin, tuntuvat poikkeukset jännittäviltä ja mielenkiintoisilta. Kauhupelissä voidaan olla koko pelin ajan kartanon sisällä. Jos kartano on oikean kaltainen ja sisältää maailmassamme normaalisti käyttäytyviä esineitä ja asioita, on yliluonnollinen elementti pelottavampaa. Kun yhden huoneen esineet ovat tavallisia esineitä, jotka toimivat odotetusti, on seuraavan huoneen pelaajaa seuraava haamumainen taulu huomattavasti pelottavampi.

3 KENTTÄSUUNNITTELUA KONEELLA

Kenttäsuunnittelu on kokonaisuuksien luomista, jolloin ison kokonaiskuvan hahmottaminen pelistä on tärkeää. Kokonaisuuksia kannattaakin paloitella pienemmiksi osiksi, joista pitää pystyä kuitenkin hahmottamaan koko pelin punainen lanka. Ensiksi olisi tärkeää määritellä pelimaailman säännöt: kuinka iso maailma on ja missä pelaaja saa liikkua. Sitten kannattaa tarkastella pelin teemaa, sillä se vaikuttaa pelin graafiseen ulkoasuun kaikkein eniten. Kauhupelissä ei seikkailla Hesburgerin pallomeressä eikä lapsille suunnatussa tasoloikkapelissä ole verenpunaista täysikuuta, missä loikkii ihmissusia.

Ennen yksittäisen huoneen tai kentän tason luomista kannattaa ensin suunnitella karkea luonnos koko kentästä (Rouse III 2001, 427). Esimerkiksi kauhupelin tapauksessa, joka sijoittuu yhteen taloon, luodaan ensin hahmotelma talosta. Sitten määritellään, mitkä huoneet kuuluvat mihinkin kenttään, minkä jälkeen vasta aloitetaan yksittäisten huoneiden suunnittelu. Luonnosten kannattaa olla ensin hyvin karkeita ja suuntaa antavia, sillä aluksi ei kannata tuhlata aikaa tarkkaan näpertelyyn. On hyvä näyttää jo hahmotelmia muille projektin jäsenille, jotta säilytetään pelin visuaalinen ilme yhtenäisenä (Rouse III 2001, 427). Kuvassa 1 on esimerkki karkeasta luonnoksesta, jota peliprojektissa on käytetty.



Kuva 1. Karkea luonnos peliprojektin kentästä.

3.1 Kuvankäsittelyohjelmat

Ennen kuin voidaan luoda peliobjekti Unityn pelimoottoriin, pitää tehdä grafiikka, jonka pohjalta pelimoottori osaa luoda peliobjektin. Kaksiulotteisessa pelissä tavallisilla kuvanmuokkausohjelmilla tulee toimeen, mutta jos haluaa aloittaa kolmiulotteisten objektien tai hahmojen luontia, tarvitaan siihen erikoistunut ohjelma. Hyviä ilmaisia kolmiulotteisien objektien luomisohjelmia ovat muun muassa Blender sekä Google Sketchup. Koska projektin pelidemo oli kaksiulotteinen, selvisi tavallisilla kuvankäsittelyohjelmilla, kuten Photoshop, hyvin.

Itselläni on jo entuudestaan tietotaitoa Photoshopista ja omistan myös kyseisen ohjelman. Photoshop on kuitenkin maksullinen. Hyviä ilmaisia vaihtoehtojakin on saatavilla. Jos Photoshop-ohjelmaan ei halua sijoittaa rahallisesti, on seuraavassa kappaleessa kerrottu hyviä ilmaisia vaihtoehtoja.

Grafiikkatyötä, jota voidaan tehdä missä vain ja joka ei tarvitse erillistä ohjelmaasennusta, on internet-selaimessa toimiva Pixlr Photo Editor. Tämä on mainio vaihtoehto Photoshopille, sillä se tarjoaa kattavan valikoiman toimintoja yksinkertaiseen kuvanmuokkaukseen. Gimp sekä Paint.net ovat myös hyviä sovelluksia kuvanmuokkaukseen, joista Gimp on melkein Photoshopin tasolla, mutta vaatii enemmän kokemusta kuin Paint.net. Photoshop-ohjelmaan tottuneet voivat vierastaa Gimp-sovelluksen ulkoasua, sillä sen toiminnot ja työkalut ovat erinimisiä ja niiden sijainti poikkeaa huomattavasti Photoshopista. Paint.net on yksinkertainen ja selkeä kuvankäsittelyohjelma, jota itse suosittelen kaikista ilmaisista vaihtoehdoista, koska sitä on helppo käyttää ja se tarjoaa tarpeeksi laajan valikoiman työkaluja sekä ominaisuuksia.

3.2 Kuvankäsittely

Grafiikkaa suunnitellessa pitää ottaa huomioon kuvan koko ja sen tiedostomuoto, grafiikan sisältö sekä visuaalinen ilme. Kaksiulotteisessa pelissä taustoja sekä muita visuaalisia elementtejä kierrätetään paljon ajan säästämiseksi. Tämä pitää ottaa huomioon jo suunnitteluvaiheessa, sillä taustagrafiikoista kannattaa tehdä sellaisia, että niitä voidaan niin sanotusti monistaa monta kertaa. Jos tekstuuri, taustagrafiikka tai peliobjekti on liian monimutkainen, se näyttää helposti kankealta ja huonolta monistettaessa. Yksittäinen objekti kuten tuoli, pöytä tai jokin muu vastaava esine tai objekti voi olla yksityiskohtainen, sillä sitä ei monisteta. Esimerkiksi seinän pala, jossa on paljon yksityiskohtia, voi näyttää hassulta, jos palasia monistaa vierekkäin. Kun seinän tausta on yksinkertainen, näyttää seinä luonnolliselta monistettaessa.

Vaikka suurin osa grafiikkatyöstä pitää tehdä itse, ei kaikkea tarvitse alusta asti kehittää. Varsinkin kaksiulotteisessa pelissä kannattaa hyödyntää jo olemassa olevaa ilmaista kuvamateriaalia. Valmiit tekstuurit sekä clip art -kuvat ovat hyviä tapoja kehittää ja parantaa omaa grafiikkaansa. Clip art -kuva on valmiiksi tehty yksittäinen piirros tietystä objektista tai esineestä. Clip art -kuvia käytetään niin Internet-selaimissa kuin työpöydän ikoneissa. Kuvia voi ostaa tai ladata ilmaiseksi omaan käyttöön (Rouse 2005). Koska piirrokset ovat yleensä yksinkertaisia, saa niistä pienellä muokkauksella hyvää yksityiskohtaa omaan peliin. Yksi sivu, josta saa hyvän määrän ilmaisia mihin käyttöön vain tarkoitettuja clip art -kuvia, on www.clker.com. Suosittelen tätä sivua, sillä olen saanut täältä hyviä ideoita ja kuvia omaan grafiikkasuunnitteluuni.

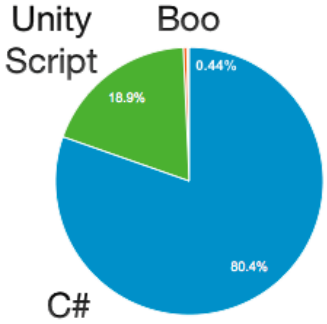
4 UNITY3D

Unity3D on Unity Technologiesin kehittämä pelimoottori, jolla voi tehdä 2D- ja 3D -pelejä eri alustoille. Unity3D tukee pelinkehitystä PC-tietokoneelle, eri konsoleille, älypuhelimille sekä internet-selaimiin. Projektissa käytetty versio Unitysta on Unity Personal 5.2.3, joka julkaistiin projektin alkuvaiheessa.

Unity3D on pelinkehitysalusta, jossa on oma käyttöliittymänsä. Pelimoottori yhdistää kuvankäsittely- ja videonmuokkaus -ohjelmista tutun käyttöliittymäympäristön sekä ohjelmointia tukevan alustan. Kaikki visuaaliset elementit ja pelimoottoriin liittyvät asetukset tehdään käyttöliittymässä. Raskas ohjelmointi suoritetaan MonoDevelop-nimisessä koodausohjelmassa. MonoDevelop tukee Unityn ohjelmointikieliä, ja se on yksinkertainen tekstieditori sekä testausalusta käyttää, jos ohjelmointi on jo tuttua.

4.1 Ohjelmointikielet

Unityn tukemat ohjelmointikielet ovat C#, UnityScript eli JavaScript sekä Boo. Kuvassa 2 on selostettu vuoden 2014 Unityn ohjelmointikielten suosiota. Unity3D versiossa 5 ja eteenpäin ei Boo -ohjelmointikielellä voi luoda uusia ohjelmakoodeja, mutta jos projekti on sisältänyt Boo -koodia, osaa Unity tunnistaa ne vielä projektin sisällä. Boo jätettiin pois uudesta Unityn versiosta resurssisyistä (Aleksandr 2014.)



Kuva 2. Unityn ohjelmointikielten käyttö (Aleksandr 2014).

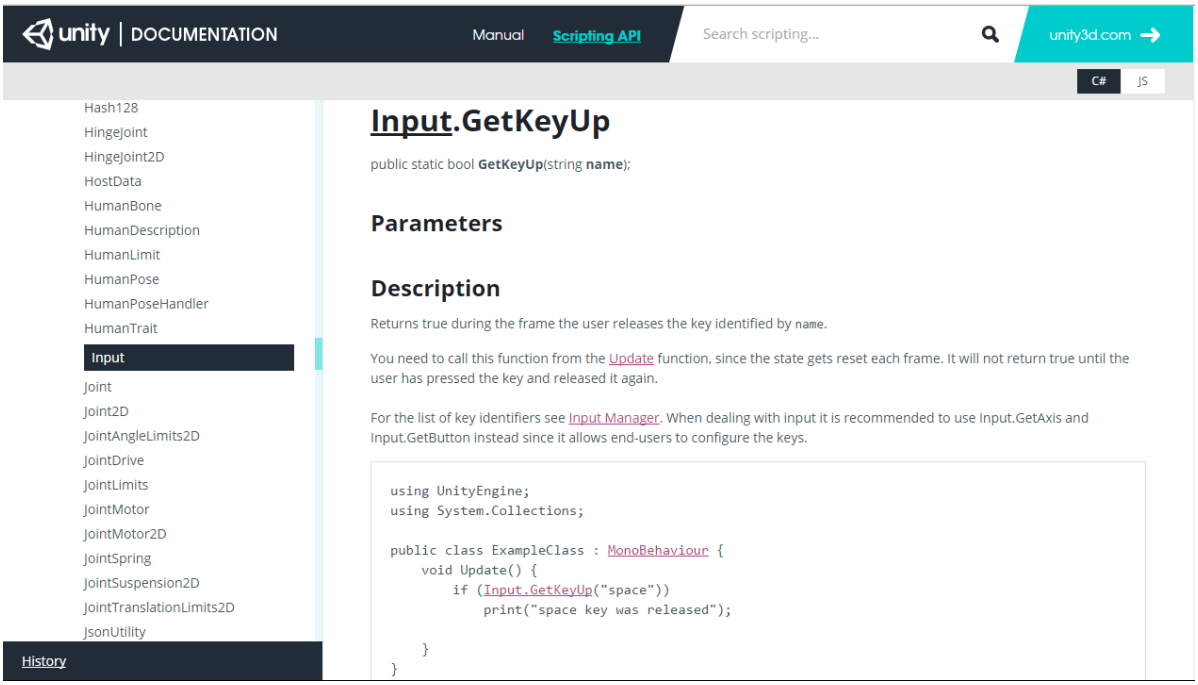
C# on yleisin Unityssa käytettävistä ohjelmointikielistä (Aleksandr 2014). C# tarjoaa monipuolisia ratkaisuja ongelmiin pelimoottorissa ja onkin hyvä ohjelmointikieli niin aloittelijoille kuin ammattilaisille. C# voi vaikuttaa hankalammalta kuin JavaScript, mutta se käyttää vain erilaista syntaksia kuin muut perinteisemmät ohjelmointikielet. Koska C# on laajalti tuettu Unityssa, on ohjeita sekä tutoriaaleja paljon C# -kielellä.

UnityScript eli JavaScript on Unityn toiseksi yleisin ohjelmointikieli (Aleksandr 2014). JavaSript on silti kaiken kaikkiaan yleisempi ohjelmointikieli kuin C# (W3Techs 2015). Jos on aikaisempaa kokemusta ohjelmoinnista, Javan opettelu on helppoa. Java on muutenkin yleinen opetuskäytössä oleva ohjelmointikieli. Javaa käytetään kuitenkin enemmän ohjelmien, kuten internet-selaimiin pohjautuvien asioiden luomiseen, joten ohjeita sekä tutoriaaleja löytyy Unityn pelimoottoriin melko niukasti.

Unity käyttää omia pelimoottoriin sidonnaisia koodikomentoja olemassa olevien koodien lisäksi. Jonkinlainen osaaminen ohjelmoinnista olisikin tärkeää, sillä kaikkia ohjelmointikielien perusasioita, kuten while ja if -lausekkeita, käytetään paljon. Pelimoottoriin uniikkia koodia on paljon, mutta Unityn kotisivuilla on paljon materiaalia ohjelmoinnista sekä kattava koodikirjasto, josta löytyy jokainen Unityn pelimoottorin koodi.

Unity Technologies on kehittänyt kotisivuilleen täydellisen koodireferenssin, josta näkee jokaisen Unityn pelimoottoriin liittyvän koodin sekä UnityScript että C# -ohjelmointikielellä. Tämän lisäksi referenssistä näkee, mihin sitä käytetään, ja sivulla on myös esimerkkitilanne, missä sitä tarvitaan (Unity 2015b). Koska C# on yleisempi ohjelmointikieli Unityssa, näyttää koodireferenssi oletuksena koodit ensin C# -kielellä.

Jos halutaan esimerkiksi tietää millainen koodi tulee toiminnolle ”paina näppäintä alas”, voidaan se etsiä koodireferenssistä. Toiminto on Input.GetKeyDown ja se toimii if -lausekkeen sisällä siten, että Input.GetKeyDown -komennon jälkeen pitää antaa näppäimistön näppäin, mitä painetaan. Tämän jälkeen if -lausekkeen sisälle tulee haluttu toiminto. Esimerkissä on käytetty print eli tulostustoimintoa, mikä tulostaa halutun tekstin valittua näppäintä painaessa (kuva 3).



Kuva 3. Kuvankaappaus Unityn koodikirjastosta (Unity 2015c).

4.2 Tutoriaalit

Tutoriaali on yksityiskohtainen opastus ja oppimisprosessi, jossa tutoriaaliin osallistujalle annetaan suoraa informaatiota ja tiettyä oppimiseen liittyvää materiaalia. Se on huomattavasti yksityiskohtaisempi ja interaktiivisempi kuin esimerkiksi luento tai kirja (Tutorial 2015). Tutoriaalissa neuvotaan tai opastetaan jokin tietty asia, eikä se yleensä kata laajempaa teoriaa.

Peliprojektissa käytettyjä tutoriaaleja olivat monet opetusvideot, jotka neuvovat Unityn pelimoottorin ja ohjelmointikomentojen käyttöä. Unity tarjoaa verkkosivuillaan sekä erityyppisten pelien tutoriaaleja että pelimoottoriin toimintoihin

perustuvia opastusvideoita (Unity 2015d). Peliprojektin tapauksessa tutoriaali on ollut opetusvideo, mutta se voi olla myös kirjallisessa muodossa. Esimerkiksi ohjekirja on eräänlainen tutoriaali. Internet-sivusto Youtube (www.youtube.fi) on hyvä sivusto opastusvideoiden löytämiseen, mutta koska Youtubeen voi kuka vain ladata videoita, pitää olla hyvin lähdekriittinen videoita katsottaessa.

4.3 Asset Store

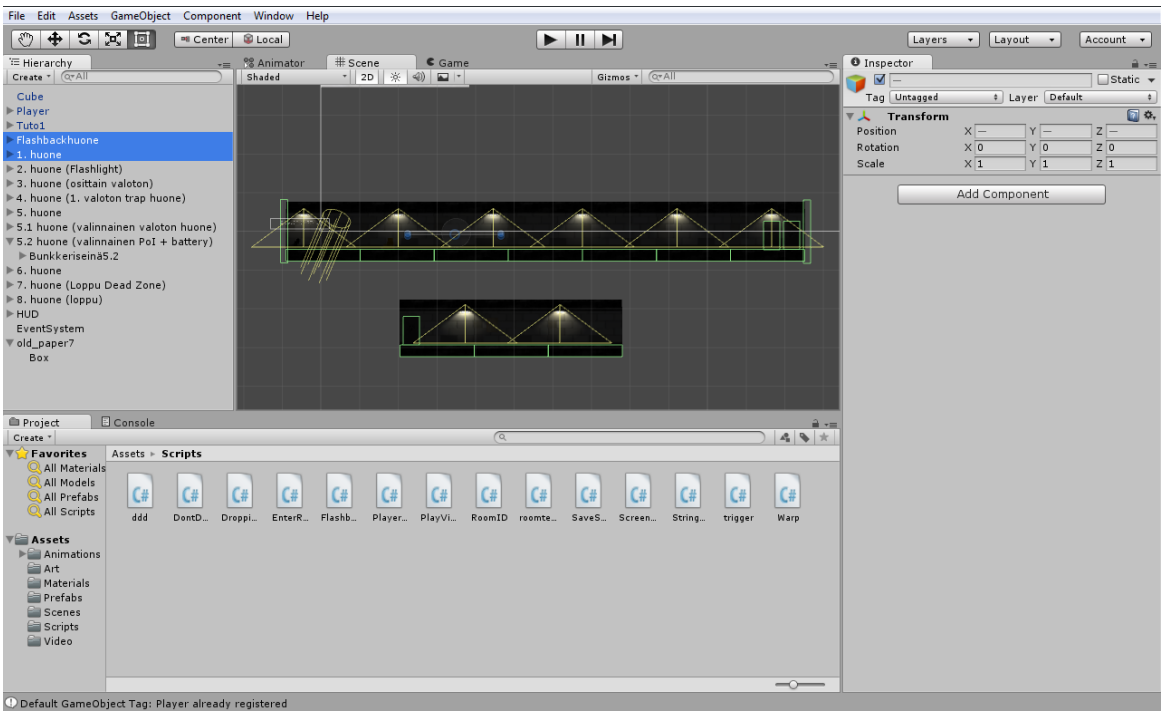
Unityn Asset Store on verkkokauppa, josta voi ostaa tai ladata ilmaiseksi Unityn pelimoottoriin tarvittavia asset-paketteja. Unityn pelimoottorissa asset tarkoittaa valmispakettia, johon voi sisältyä muun muassa peliobjekteja, tekstuureita, kuvia tai kolmiulotteisia grafiikkamalleja. Asset Storesta saa myös ostettua pelimoottoriin valmiita komponentteja ja modifikaatioita, jotka auttavat saavuttamaan tietyn lopputuloksen pelissä. Verkkokaupan valmispaketit auttavat ihmisiä, jotka eivät halua opetella ohjelmoimaan tai eivät osaa sitä hyvin, sillä verkkokauppa on täynnä erilaisia muokkausohjelmia kenttien tekemisestä aina hahmojen yksityiskohtaiseen luontiin.

Kirjoitushetkellä (23.11.2015) suosiossa olivat fysiikkamoottoria muokkaavat lisäosat sekä valmiit mallinnukset ja pelimekaniikkamodifikaatiot. Suurin osa monimutkaisista editoreista sekä valmispaketeista ovat maksullisia, sillä ne vähentävät koodaustyötä ja antavat samalla ammattimaista näkyvyyttä peliin. Suosituimmat ilmaiset valmispaketit olivat lähinnä kolmiulotteisia objekteja, avatar -hahmoja tai erilaisia vihollishahmoja, sillä niiden tekemiseen ei vaadita paljon osaamista. Monet omaksi iloksi pelejä tekevät ihmiset laittavat niitä ilmaiseksi muille jaettavaksi (Unity 2015a.)

Peliprojektissa ei ole käytetty ainuttakaan valmispakettia verkkokaupasta, koska peliä yritettiin tehdä mahdollisimman paljon ilman rahallista sijoitusta. Oman grafiikkatyylin säilyttämiseksi kaikki peliobjektit tehtiin itse.

5 PELIKENTÄN LUONTI UNITYN PELIMOOTTORIIN

Unity käyttää kuvan- sekä videonmuokkausohjelmista tuttua visuaalista editoria, jonka avulla pystyy toteuttamaan ison osan peliin sisältyvistä osioista. Kaikki visuaaliset ja fysiikkamoottorin palasiin liittyvät lisäykset tehdään Unityn omaa editoria käyttäen. Ohjelmointi on myös isossa osassa, sillä kaikki tarkemmat toiminnot pitää ohjelmoida peliin itse. Kaikki pelimekaniikkaan liittyvät asiat ovat joko ohjelmointia tai yhdistelmä Unityn valmiita toimintoja, joihin liitetään oma ohjelmakoodi. Unity tukee kolmea ohjelmointikieltä: C#, UnityScript eli JavaScript sekä Boo. Peliprojektissa on käytetty C#-ohjelmointikieltä, sillä se on yleisin Unityssa käytetty ohjelmointikieli (Aleksandr 2014.)



Kuva 4. Unityn pelinmuokkausnäkymä.

Koska Unitylla voi luoda kaksi- sekä kolmiulotteisia pelejä, on Unityssa kaksi eri fysiikkamoottoria. Pelin tekeminen ja toimintojen funktio on erilaisempi fysiikkamoottorien toiminnasta riippuen. Huomattava ero on siinä, miten peliobjekteja sijoitellaan x-, y-, ja z-akselilla. Kaksiulotteisessa pelissä peliobjekteilla on vain kaksi akselia: yksinkertainen rotaatio myötä- tai vastapäivään sekä järjestys, mikä objekti näkyy päällimmäisenä tai alimmaisena. Suurin ero kaksiulotteisen

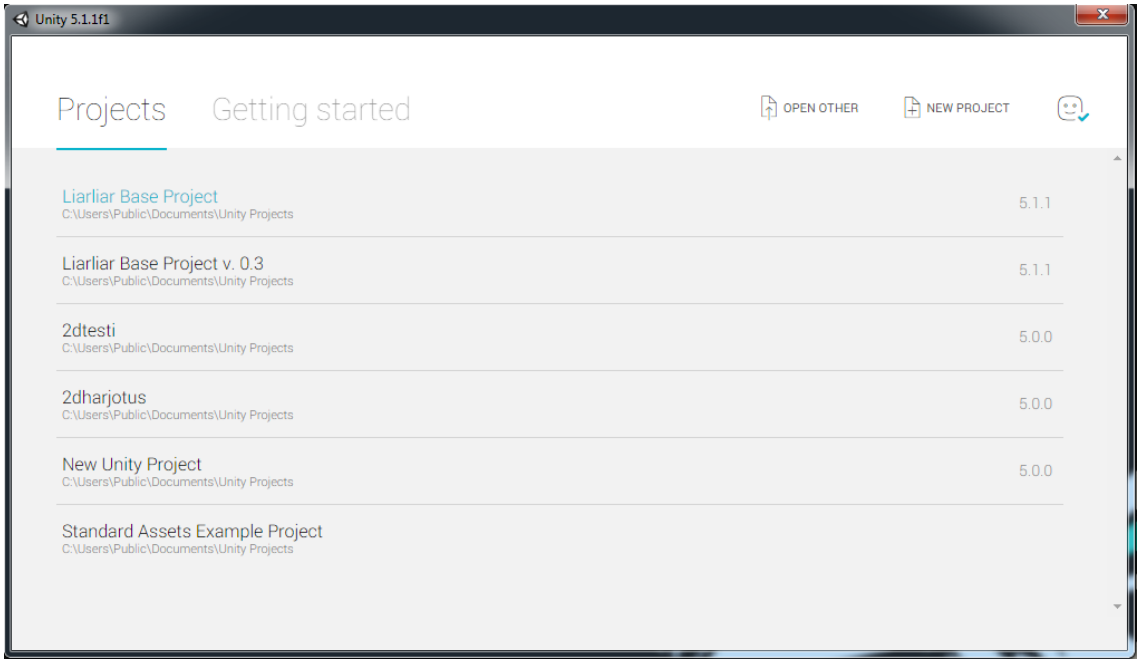
ja kolmiulotteisen pelin välillä on ohjelmointi. Tiettyjä toimintoja pitää koodata eri tavalla pelin kategoriasta riippuen.

Tässä peliprojektissa tullaan kertomaan pelkästään kaksiulotteisen pelimoottorin toiminnoista. Jos tekstissä ei erikseen mainita, tarkoitetaan silti Unityn kaksiulotteista pelimoottoria. Opinnäytetyössä mainitut Unityn peliprojektin esimerkit, ohjeet sekä ohjelmakoodit ovat tarkoitettu 2D-peliin.

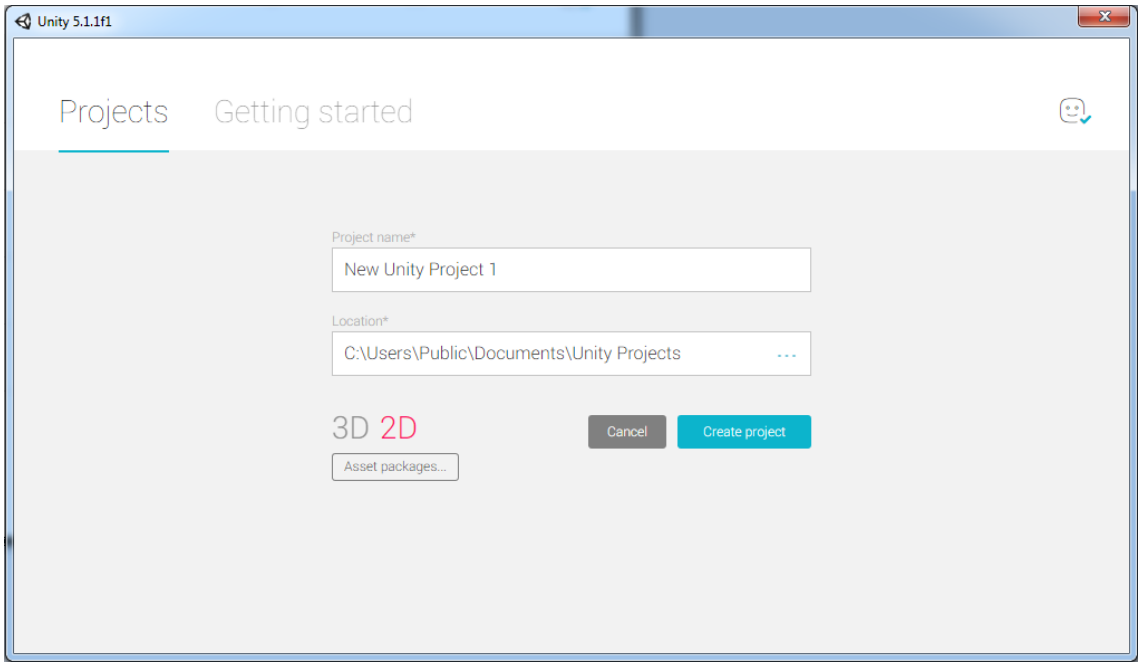
5.1 Projektin aloittaminen

Kun pelin tekeminen aloitetaan Unityssa, kutsutaan sitä projektiksi. Projekti on keskeneräinen peli ja vasta kun kehittäjät ovat tyytyväisiä, voidaan peli julkaista Unityssa ja kutsua sitä peliksi. Unityn projektia ei saa sekoittaa peliprojektiin, joka on taas koko pelin tekemiseen liittyvä prosessi aina ideoinnista pelin julkaisupäivään asti. Unityn projekti on puolestaan käytännössä pelin kehittämistä visuaaliseksi ja pelimekaaniseksi kokonaisuudeksi. Suurin osa peliprojektia on kuitenkin pelin muokkaamista ja rakentamista Unityssa.

On tärkeää jo projektia aloittaessa tietää, onko peli kokonaan kaksiulotteinen vai onko siinä kolmiulotteisia elementtejä, sillä alun projektin luomisnäkymässä (kuva 6) voi valita vaihtoehdot 2D tai 3D. 3D on kolmiulotteinen versio Unityn pelieditorista ja sillä pystyy tekemään myös kaksiulotteisia pelejä. 2D on taas tarkoitettu pelkästään kaksiulotteisten pelien tekoon eikä sillä voi tehdä mitään muuta. Peliprojekti on tehty 2D -editorissa, sillä se on kokonaan kaksiulotteinen, eikä projektiin haluttu turhia kolmiulotteisia toimintoja pelinmuokkausnäkymään. Kuvassa 5 on aloitusnäkymä, josta voidaan luoda uusi projekti tai tarkastella olemassa olevaa projektia.



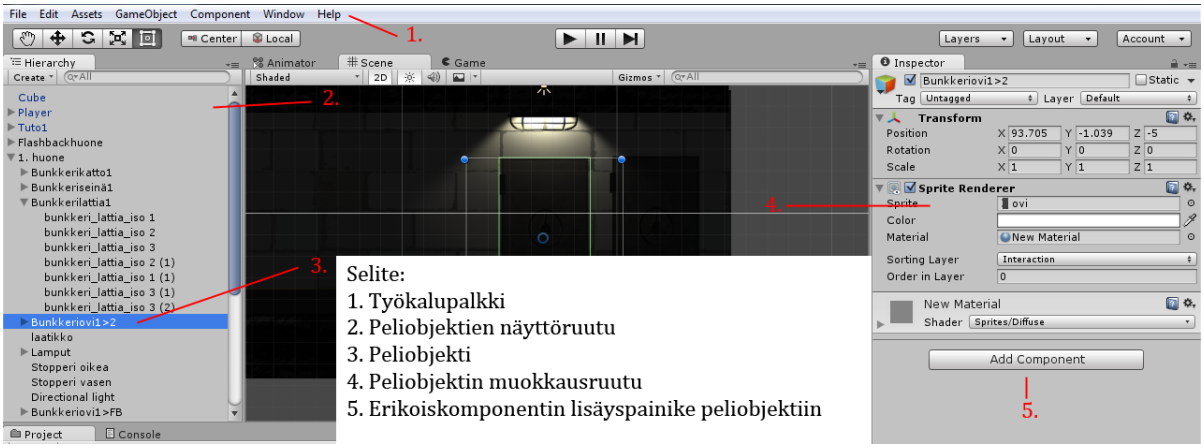
Kuva 5. Projektin aloitusnäkymä.



Kuva 6. Projektin luontinäkymä.

5.2 Peliobjektit

Peliobjekti on tärkein muokattava asia koko Unityn pelimoottorissa. Kaikki peliin liittyvät toiminnot ja mekaniikat ovat kytköksissä peliobjekteihin, sillä ne ovat asioita, jotka näkyvät ja toimivat itse pelissä. Peliobjektiin voi kuulua kuva, video, ääni, erikoiskomponentti, kamera, ohjelmakoodi tai animaatio. Peliobjekti voi olla myös tyhjä, jos haluaa käyttää sitä muiden peliobjektien lajitteluun. Peliobjektin sisällä voi olla toinen peliobjekti ja ne voivat olla linkitettynä toisiinsa (kuva 7).

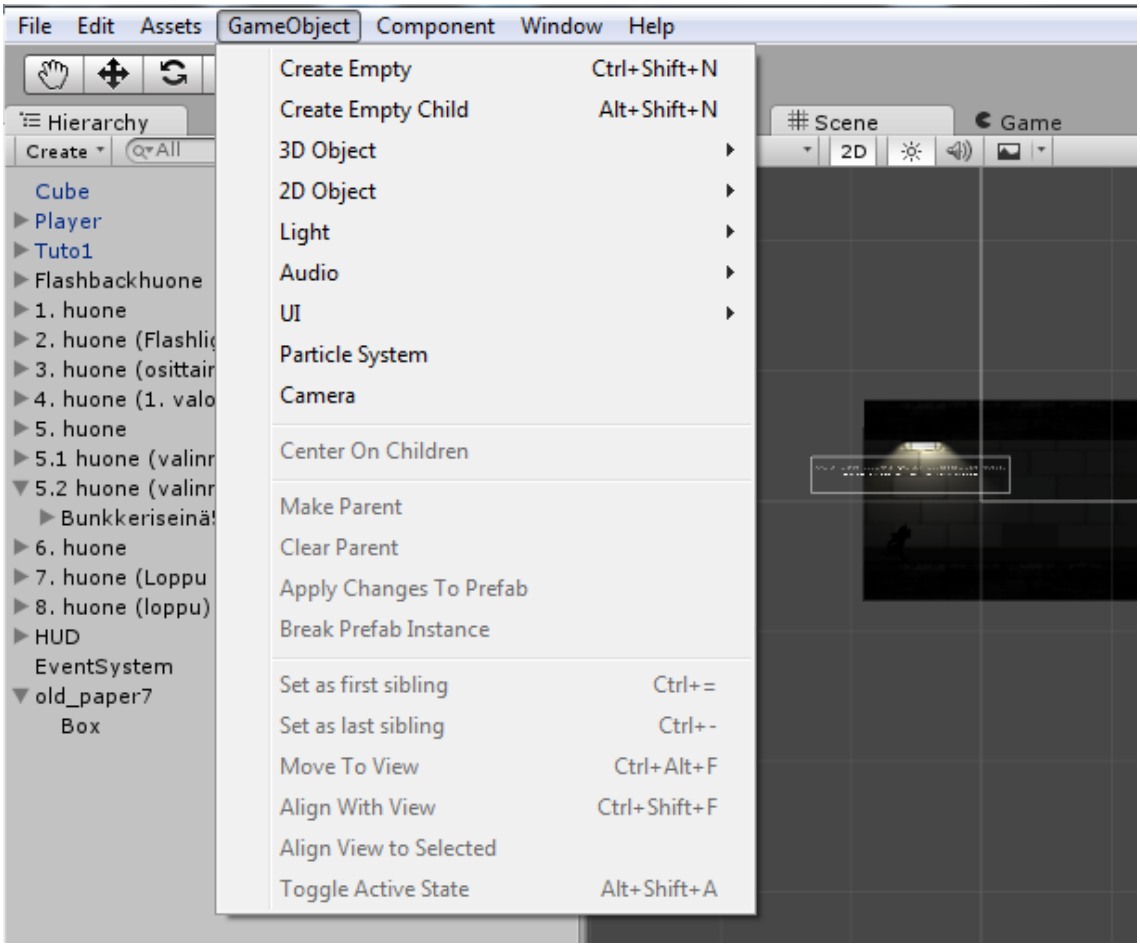


Kuva 7. Peliobjekteihin liittyvät työkalut ja toiminnot.

Peliobjekteja lisätään kohdasta GameObject, jonka alla on monta eri vaihtoehtoa. Ne on luokiteltu eri kategorioihin niiden sisältöjen ja toimintojen mukaan. Peliobjekteja voidaan myös siirtää suoraan muokkausnäkymään käyttämällä siirrä ja raahaa -toimintoa. Kuvassa 8 on tarkemmin näytetty GameObject-kohta ja taulukossa 1 on listattu peliobjekteja, joita voidaan luoda GameObject-valikosta.

Taulukko 1. Lista peliobjekteista GameObject-valikossa.

Create Empty	Luo tyhjän peliobjektin
Create Empty Child	Luo tyhjän peliobjektin valitun peliobjektin alle
3D Object	Tämän valikon alta voi luoda erimuotoisia kolmiulotteisia esineitä tai kappaleita
2D Object	Tämän valikon alta voi luoda vain sprite -grafiikan eli kaksiulotteisen kuvan
Light	Valikosta löytyy lajitelma eri valaisumenetelmiä
Audio	Vaihtoehto luoda äänilähde peliin
UI	Voidaan luoda käyttöliittymään kuuluvia objekteja esimerkiksi painike tai teksti
Camera	Pelikamera



Kuva 8. GameObject -valikko.

5.3 Ohjelmointi

Suuri osa toiminnoista pitää ohjelmoida peliin itse. Unitylla on omat komentonsa eri toimintojen ja tapahtumien kutsumiseen, mutta se noudattaa sen ohjelmointikielen sääntöjä, mikä on valittu. Jos on kokemusta C# -kielestä, on ohjelmointi kyseisellä kielellä samanlaista Unityssa. Unityn mukana tulee ohjelma, MonoDevelop, jolla kirjoitetaan pelin ohjelmakoodit. MonoDevelop on hyvin monipuolinen ohjelmointialusta, joka tukee Unityn ohjelmointikieliä. Siihen sisältyy myös täydellinen koodikirjasto, josta löytyy jokainen Unityssa tarvittava komento.

Ennen kuin ohjelmakoodit toimivat pelissä, pitää ne lisätä peliobjektiin. Yksikään luotu koodi ei toimi, jos sitä ei yhdistä peliobjektiin. Peliobjekti voi olla täysin näkymätön, jos halutaan koodata jotain koko peliin vaikuttavaa, esimerkiksi tallennus ja lataustoiminto. Ohjelmakoodin voi lisätä myös näkyvään objektiin, esimerkiksi pelaajahahmoon (Abhinav 2014.)

Esimerkki:

Halutaan luoda ohjelmakoodin, joka mahdollistaa pelaajahahmon yksinkertaisen liikkumisen vasemmalle ja oikealle. Ensin pitää luoda uusi koodi valitsemalla Assets, sitten Create ja lopuksi C# Script. Ohjelmakoodin nimeksi kirjoitetaan Playercontroller (nimeä käytetty omassa projektissa) ja se avataan MonoDevelop-ohjelmassa. On tärkeää kirjoittaa samat komennot kuin kuvassa 9 ja tallentaa ohjelmakoodi. Koodi tallennetaan ja se on nyt näkyvissä Unity-projektissa.

Tämän jälkeen pitää ohjelmakoodi lisätä peliobjektiin, joka on tässä tapauksessa pelaajahahmo. Kun peliobjekti on valittuna, voidaan objektin muokkausnäkymästä valita Add Component, sitten Scripts ja viimeiseksi valitaan luotu koodi Playercontroller. Pelaajahahmossa on koodi, joka mahdollistaa liikkumisen vasemmalle ja oikealle. Jokainen ohjelmakoodi lisätään samalla tavalla Unityyn, ja ainoastaan koodin sisältö ja mihin se on liitetty määrittää toiminnon.

Playercontroller.cs | trigger.cs | Save

Playercontroller ▸ No selection

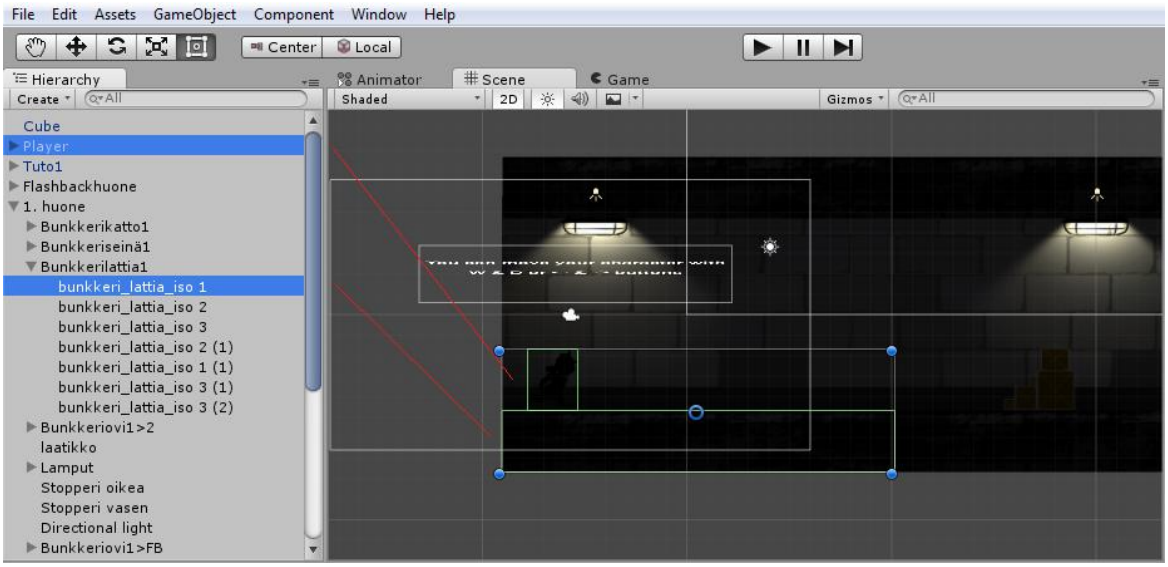
```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Playercontroller : MonoBehaviour {
5  public float speed = 50f;
6      private Rigidbody2D rb2d;
7
8      void Start ()       {
9
10         rb2d = gameObject.GetComponent<Rigidbody2D>();
11     }
12
13     void Update ()       {
14
15     }
16
17     void FixedUpdate()      {
18
19         float h = Input.GetAxis ("Horizontal");
20         rb2d.AddForce ((Vector2.right * speed) * h);
21     }
22
23 }
```

Kuva 9. Ohjelmakoodi pelaajahahmon liikuttamiseen (GucioDevs 2015).

5.4 Box collider sekä trigger

Collider on tietyn muotoinen rajaava alue, jolla määritetään rajapinta, mistä objektit eivät pääse läpi. Tämän avulla saadaan luotua maailma, jossa hahmot ja esineet pysyvät paikoillaan. Collider-toiminnolla luodaan kenttään alue, joka mallintaa oikean elämän rajoja. Toiminnon avulla voidaan esimerkiksi tehdä peliobjektin kuvaan rajapinta, josta pelaajahahmo ei pääse läpi. Näin on luotu fyysinen este hahmolle. Collider-toiminnolla voidaan myös luoda lattia, johon määritellään fysiikkamoottorista painovoima. Tällä tavoin pelaajahahmo tai muut objektit, jotka painovoiman alaisena ovat, eivät putoa lattian läpi. Jos objektista jättää collider-toiminnon pois, tarkoittaa se sitä, että objektia voidaan käyttää taustalla. Esimerkiksi taustaseinässä ei ole collider-toimintoa kaksiulotteisessa pelissä, sillä pelaajan ei haluta törmäävän taustaan vaan kävelevän sen ohitse.

Esimerkkinä collider-toiminnosta on kuva 10. Pelikenttään on luotu lattia, jossa on suorakulmainen collider sekä pelaajahahmo, josta collider myös löytyy. Pelaajahahmoon on luotu painovoima, mutta collider-toiminnon ansiosta hahmo ei tipu loputtomiin alaspäin, vaan pysähtyy lattiaan.

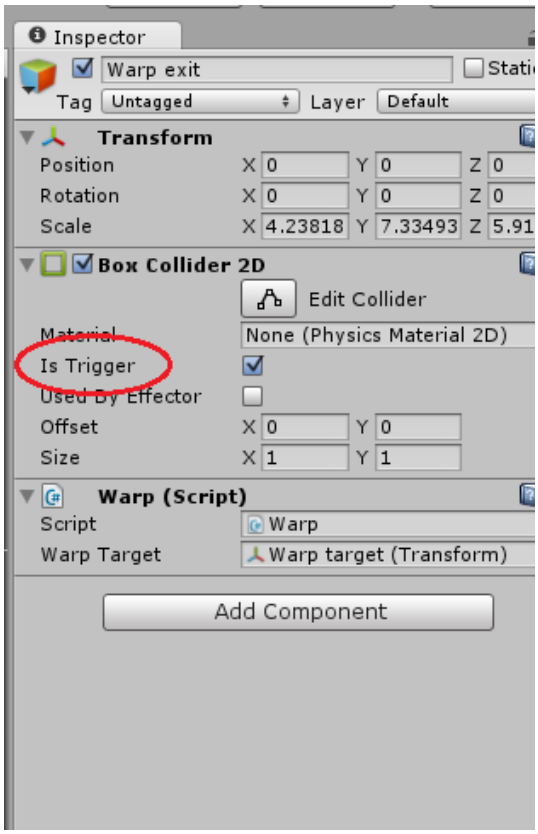


Kuva 10. Esimerkki box collider-toiminnoista peliprojektissa, jotka näkyvät vihreinä laatikoina.

Collider-toimintoja on kaksiulotteisessa fysiikkamoottorissa neljä: Box, Circle, Edge ja Polygon. Box collider on suorakulmion muotoinen, joka tekee siitä täydellisen vaihtoehdon esineisiin, lattioihin, kattoihin ja muihin rakennelmiin sekä muodostelmiin. Koska kaksiulotteisien pelien kentissä on eniten suorakulmaisia objekteja, on box collider suosituin collider-tyypeistä. Circle collider on ympyrän muotoinen ja soveltuu hyvin palloihin ja muihin pyöreisiin objekteihin esimerkiksi renkaisiin. Edge collider on puolestaan viiva, josta voidaan tehdä ääriviivoja niin sanottujen ankkuripisteiden ansiosta. Ankkuripiste on kohta, joka mahdollistaa viivan suunnan vaihtamisen. Polygon collider -toiminnossa on valmis viisikulmion muotoinen alue, joka koostuu viidestä ankkuripisteestä. Kyseinen collider ei ole rajoitettu viisikulmion muotoon, sillä ankkuripisteitä voi siirtää miten haluaa.

5.5 Ontrigger2D

Collider-toimintoa voidaan myös käyttää muuhunkin kuin fyysisten rajojen luomiseen. Toiminnon muokkausruudusta löytyy komento ”Is Trigger” (kuva 11), joka muuttaa collider-toiminnon niin sanotuksi triggeriksi eli eräänlaiseksi laukaisimeksi, jolla voidaan aktivoida tiettyjä tapahtumia tai toimintoja. Trigger-toiminto eli laukaisin muuttaa collider-toiminnon muotoa siten, että rajapinta ei enää estä muita objekteja kulkemasta sen läpi, vaan kirjaakin ylös aina kun jokin objekti liikkuu sen sisällä.



Kuva 11. Box collider-toiminnon muuttaminen triggeriksi eli laukaisimeksi.

Laukaisinta voidaan kutsua kolmella eri komennolla: OnTriggerEnter2D, OnTriggerExit2D sekä OnTriggerStay2D. Jokainen OnTrigger-komento tallentaa muistiin objektin liikkumisen collider-toiminnossa. Komennoilla aktivoidaan tapahtumia sen perusteella, mitä objekti teki collider-toiminnon sisällä. OnTriggerEnter2D tallentaa objektin sisääntulon, OnTriggerExit2D objektin lähdön ja OnTriggerStay2D laskee kuinka kauan objekti on collider-toiminnon sisällä.

OnTrigger-komentoja käytetään seuraavasti:

```
void OnTriggerEnter2D (Collider2D other) {
        // Tulostaa ”Tervetuloa” kun peliobjekti saapuu colliderin sisälle
        print("Tervetuloa");
        }

void OnTriggerExit2D (Collider2D other) {
        // Tulostaa “Tervemenoa” kun peliobjekti lähtee colliderista
        print(”Tervemenoa”);
        }

void OnTriggerStay2D (Collider2D other) {
        // Tulostaa ”Hei!” niin kauan kuin peliobjekti on colliderin sisällä
        print(”Hei!”);
        }
```

Pelissä on toiminto, jolla saadaan tekstiä näkyviin, kun pelaaja kävelee tiettyyn kohtaan. Tämä on toteutettu laukaisinta käyttäen. Koko ohjelmakoodi löytyy liitteestä yksi, ja siihen viitataan tässä osiossa.

Ohjelmakoodissa on käytetty OnTriggerEnter2D ja OnTriggerExit2D -komentoja määrittämään onko tekstitoiminto tosi vai epätosi. Koska laukaisin tietää, milloin objekti saapuu tai lähtee rajatusta collider-toiminnosta, on sitä helppo käyttää määrittelemään pelaajan sijainti. Koodiin on myös määritelty muuttujia, jotka luovat määritelmiä koodin sisälle. Tärkein muuttuja on ”teksti”, joka on koodin ulkopuolinen tekijä. Muuttuja ”teksti” voidaan määritellä Unityssa ja se on valmiiksi kirjoitettu tekstiosio.

Funktion, ColorChange(), sisällä määritellään tekstin väri sekä näkyvyys. Tällä funktiolla saadaan teksti näkymättömäksi. Jos OnTrigger-komennon tulos on funktiossa epätosi, on teksti näkymätön. Jos OnTrigger-komento antaa tuloksen tosi, on teksti puolestaan näkyvissä. Näin saadaan ilmestyvä ja katoava teksti, kun pelaaja kävelee tietyn alueen ohi (Xenosmash Games 2014; liite 1.)

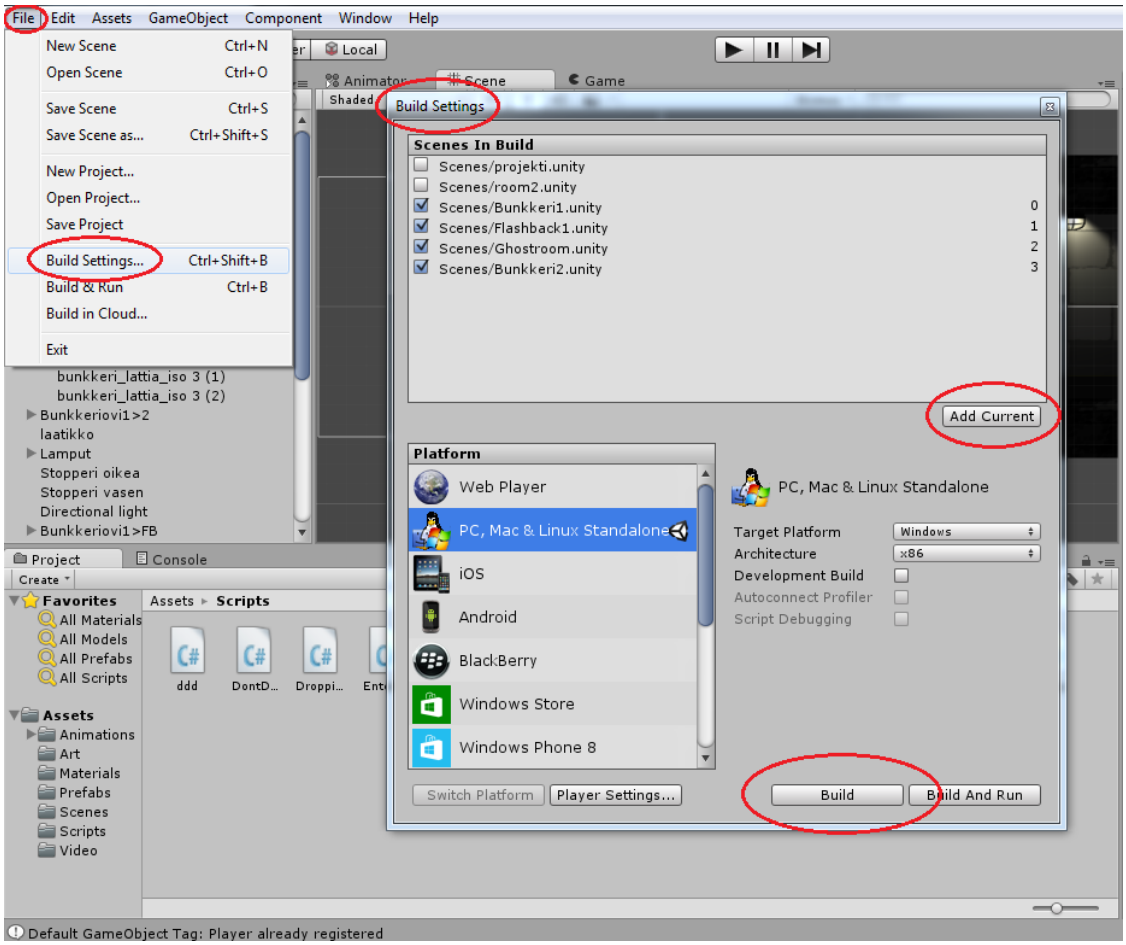
5.6 Siirtymät ja kohtaukset

Kaksiulotteisessa pelissä on kaksi helppoa tapaa siirtyä eri tasojen sekä eri kenttien välillä. Ensimmäinen tapa on tehdä jokaisesta tasosta oma scenensä eli kohtauksensa. Kohtaus on oma muokattava osionsa, joka on erillinen muista projektin kohtauksista. Niillä on hyvä erotella eri alueita pelistä. Esimerkiksi yksi taso pelistä on oma kohtauksensa ja päävalikko omansa. Koska peleihin tulee yleensä paljon eri alueita, toimintoja, välivideoita ja muita pelimekaanisia toimintoja, olisi mahdotonta saada kaikki toimimaan yhden muokkausnäkymän alla.

Jos tehdään jokaisesta tasosta oma kohtauksensa, pitää olla jokin tapa siirtyä seuraavan kohtaukseen tason loputtua. Peliprojektissa on käytetty ovea, jota painamalla latautuu seuraava kohtaus. Tähän tarvitaan muutamaa eri komentoa. Ensin pitää OnTriggerStay2D -komennolla määritellä, milloin pelaaja on oven kohdalla, jonka jälkeen todetaan Input.GetKeyUp -komennolla, mitä painiketta pitää painaa, jotta seuraava kohtaus latautuisi. Lopuksi käytetään Application.LoadLevel -komentoa, joka suorittaa itse kohtauksen vaihdon. Itse ohjelmointi voidaan toteuttaa seuraavalla tavalla:

```
void OnTriggerStay2D (Collider2D other){
        // ”e” näppäintä painaessa ladataan haluttu scene
        if(Input.GetKeyUp("e"))
                Application.LoadLevel (Scenen nimi);
}
```

Jotta Application.LoadLevel toimisi, pitää Unityssa määritellä kohtauksille Build Load eli kokoamisjärjestys. Tämä tapahtuu valitsemalla Unityssa ensin File ja sitten Build Settings (kuva 12). Aluksi Build Settings -osio näyttää tyhjältä, mutta valitsemalla Add Current, voidaan lisätä nykyinen kohtaus kokoamisjärjestykseen. Kuvassa 12 näkyy pelidemossa olevat kentät sekä niiden järjestys. Kuvassa näkyy myös käyttämättömiä kohtauksia. Kaikki kohtaukset, joita ei ole merkitty sinisellä aktiiviseksi jäävät pelin ulkopuolelle, kun se ladataan valmiiksi. Painamalla Build-toimintoa, Unity lataa kaikki aktiiviset kohtaukset ja muodostaa näistä pelikokonaisuuden. Vasta kun kyseinen kohta on tehty, pystytään Application.LoadLevel -komennolla siirtymään kohtauksesta toiseen.



Kuva 12. Pelidemon Build Load eli kokoamisjärjestys.

Toinen tapa on luoda koko kentästä yksi kohtaus ja liikuttaa pelaaja eri tasoihin. Tasot luodaan editoriin, mutta ne pidetään erillään, jotta pelaajan siirtyminen näyttää osuvalta ja peliin kuuluvalta. Peliprojektissa yksi kenttä muodostuu monesta eri huoneesta, joista jokainen toimii omana tasonaan. Kenttä on bunkkeri ja huoneet ovat kentän tasoja.

Samalla tavalla kuin ensimmäisessä metodissa, pitää ensin luoda jokin, jolla siirrytään seuraavaan tasoon. Peliprojektissa on taas käytetty ovea, jota painamalla pelaajan hahmo siirretään haluttuun tasoon. Tämän lisäksi transform.position -komennolla saa siirrettyä peliobjektin haluttuun paikkaan x- ja y-akselilla. Koodin voi toteuttaa seuraavalla tavalla:

```
void OnTriggerStay2D (Collider2D other) {
        if (Input.GetKeyUp ("e")) {
        // classin alkuun pitää luoda muuttuja warpTarget, jolloin voidaan määritellä myö-
        hemmin alue, johon peliobjekti siirretään
                other.gameObject.transform.position = warpTarget.position;
        }
}
```

Unitysta voidaan erikseen määritellä kohde, johon peliobjekti siirretään, kun ”e” -näppäintä on painettu. Muuttuja ”warpTarget” määrittelee sijainnin, johon peliobjekti siirretään. Tämä muuttuja pitää luoda ohjelmakoodin alkuun, jotta sitä voidaan määritellä myöhemmin Unityssa (rm2kdev 2015; liite 2.)

6 YHTEENVETO

Vastaus opinnäytetyössä esitettyyn kysymykseen, onko mahdollista luoda videopeli Unityn pelimoottoria käyttäen ilman aikaisempaa kokemusta tai ammattitaitoa, on kyllä. Sen pystyy myös toteuttamaan käytännössä ilmaiseksi. Internetistä löytyy ilmaisia materiaaleja sekä ohjelmia, joilla pelin pystyy toteuttamaan ilman taloudellisia panostuksia. Jos ilmaista graafista materiaalia lainaa Internetistä, se näkyy myös pelin ulkoasussa, jolloin pelin on vaikea erottua joukosta. Tästä syystä graafinen ulkoasu on tehty pelidemoon itse.

Valitettavasti mitä enemmän tekee itse tai hyödyntää ilmaisia ohjelmia sekä materiaaleja sitä kauemmin projekti tulee kestämään. Sen lisäksi, että pitäisi opetella Unityn pelimoottorin käyttöä, pitää tutustua pahimmassa tapauksessa useaan uuteen ohjelmaan. Pelkästään pelimoottorin opetteleminen vie aikaa, sillä pelimoottorissa yhdistyy ohjelmointikielten opettelua sekä visuaalisen muokkausohjelman käyttö. Jos tähän pitää lisätä vielä yksikin uusi ilmaisohjelma, on opettelemisen määrä suuri. Onneksi itselläni oli jo entuudestaan kokemusta Photoshopin käytöstä, joten riitti, että opiskelin vain Unityn käyttöä.

Unityn pelimoottori oli selkeästi projektin eniten aikaa vievä osio. Jo pelkkä pelikentän luominen oli työlästä, mutta samalla joutui opettelemaan jatkuvasti uutta asiaa. Tämä hidasti projektia valtavasti, sillä ensin piti opetella miten tiettyä asiaa voidaan käyttää Unityssa, sitten samaa asiaa piti vielä soveltaa omaan näkemykseen sopivaksi.

Myös itse ohjelmointi hidasti projektia. Käytännössä Unityn pelimoottorissa jokainen komento on uusi ja pelkästään siinä toimiva, joten opeteltavana oli melkein uusi ohjelmointikieli. Koska komentoja ei ole missään muualla, ovat ongelmatilanteet hankalia ja hyvin vaikeita korjata. Esimerkiksi opastusvideon mukaan tallennus ja lataus -toiminnon olisi pitänyt toimia, mutta ongelmaksi muodostui tietojen lataaminen halutussa järjestyksessä. Tästä syystä toiminto jätettiin demosta pois ja osa koodista säilytettiin käytettäväksi muualla.

Projektin hitaasta etenemisestä huolimatta se ei koskaan ollut mahdotonta. Unityn kotisivuilla on erittäin laaja arkisto opetusvideoita, joita seuraamalla oppii tärkeimpiä asioita Unitysta. Sivuilla on myös pelidemoista esimerkkejä, joista saa hyvän käsityksen miten tietyn kategorian peli tulee toteuttaa. Myös Youtube-sivustolta löytyy paljon opetusmateriaalia erityyppisten pelien tekemiseen. Videot ovat yleensä yksittäisen henkilön tekemiä ja perustuvat omiin kokemuksiin, joten niitä pitää tarkastella hyvin lähdekriittisesti.

Pelikentän teossa jonkin kuvankäsittelyohjelman osaaminen on ensisijaisen tärkeää. Jokainen graafinen osa pitää tehdä itse, joten kuvankäsittelyohjelman osaaminen on pakollista, jos ei halua turvautua valmiiseen materiaaliin. Piirtotaito ei ole pakollista, mutta auttaa huomattavasti, sillä jotain grafiikkaa saattaa joutua piirtämäänkin. Itse sain kaiken tehtyä kuvankäsittelyohjelmalla, koska kaksiulotteisessa pelissä graafinen ulkoasu ei ole liian monimutkainen. Vaikka kaksiulotteisen grafiikan tekeminen on helpompaa kuin kolmiulotteisen, on kuvankäsittely silti työlästä. Omaan projektiin tuli asetettua aluksi liian isot vaatimukset graafiselle ulkoasulle, joten niiden tekeminen oli aikaa vievää.

Opinnäytetyötä varten oli tarkoitus tehdä pelkkä pelidemo, mutta peliä on suunniteltu paljon pidemmälle. Projektia olisi tarkoitus jatkaa, jotta saataisiin aikaiseksi valmis markkinoitava peli. Jatkotutkimusta opinnäytetyön aiheelle olisikin markkinoinnissa sekä pelien kannattavuudessa. Olisi myös mielenkiintoista nähdä, miten Unitylla toteutettaisiin maksullista materiaalia hyödyntäen peli ja miten se selviäisi pienemmän budjetin pelejä vastaan.

Vaikka tällä hetkellä (vuosi 2015) itse toteutettuja pelejä löytyy runsaasti, eikä suurimmalla osalla itsenäisillä pelinkehittäjillä mene hyvin (Cliff 2015), suosittelen silti tutustumaan Unityn pelimoottoriin, jos pelinkehitys vähäänkään kiinnostaa. Unity on täysin ilmainen ja tarjoaa laajan valikoiman työkaluja ja resursseja oppimiseen. Pelimoottorissa on haastetta niin aloittelijoille kuin ammattilaisillekin ja tarjoaa mielenkiintoisen ympäristön ohjelmoinnin opetteluun. Unityn ison markkinaosuuden sekä laajan käyttäjäkunnan ansiosta (Unity 2015b), on todennäköistä, että Unity tarjoaa jatkossakin pelikehityspalveluitaan ja auttaa monia itsenäisiä pelinkehittäjiä eteenpäin.

Peliala on alati kasvava ja ohjelmat kuten Steam sekä varainkeruusivustot (IndeGoGo ja KickStarter) auttavat itsenäisiäkin pelinkehittäjiä saamaan rahoitusta peleilleen ja näkyvyyttä markkinoille. Silti viime vuosina on ollut havaittavissa itsenäisten pelinkehittäjien räjähtänyt kasvu, mikä on hankaloittanut uusien pelinkehittäjien selviämistä isojen pelistudioiden rinnalla (Cliff 2015). Vaikka itsenäisillä pelinkehittäjillä menee huonommin kuin aikaisemmin, on opinnäytetyöni aihe silti ajankohtainen.

LÄHTEET

Abhinav a.k.a Demkeys. 2014. Unity Tutorial: Creating Game Objects and Adding Components via Script. Viitattu 16.12.2015 https://www.youtube.com/watch?v=_S0xFbW2UdQ.

Aleksandr. 2014. Documentation, unity scripting languages and you. Viitattu 20.11.2015 http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/.

Bates, B. 2004. Game Design, Second Edition. Boston: Thomson Course Technology PTR.

Cliff, H. 2015. Causes of the indie Apocalypse (maybe). Viitattu 11.12.2015 http://positech.co.uk/cliffsblog/2015/09/19/causes-of-the-indie-apocalypse-maybe/.

C Sharp Accent Tutorials. 2015. Unity 5 Tutorial Save System. Viitattu 9.12.2015 https://www.youtube.com/watch?v=dRDIiK7AGn4.

GucioDevs. 2015. Unity 5 2D Platformer Tutorial - Part 2 - Movement, Animation. Viitattu 10.12.2015. https://www.youtube.com/watch?v=4hIVBYgXnFQ.

rm2kdev. 2015. 2D RPG Warping - Unity3D. Viitattu 9.12.2015 https://www.youtube.com/watch?v=Ra01vCUG4-o&list=PL_4rJ_acBNMH3SExL3yIOzaqj5IP5CJLC&index=6.

Rogers, S. 2010. Level Up! The Guide To Great Video Game Design. Chichester: John Wiley & Sons, Ltd.

Rouse III, R. 2001. Game Design: Theory and Practice. Teksas: Wordware Publishing, Inc.

Rouse, M. 2005. Clip art. Viitattu 16.12.2015 http://whatis.techtarget.com/definition/clip-art.

Tutorial. 2015. Wikipedia. Viitattu 10.12.2015 https://en.wikipedia.org/wiki/Tutorial.

Unity. 2015a. Asset Store. Viitattu 23.11.2015 https://www.assetstore.unity3d.com/en/.

Unity. 2015b. Company Facts. Viitattu 12.12.2015 https://unity3d.com/public-relations.

Unity. 2015c. Scripting Reference. Viitattu 25.11.2015 http://docs.unity3d.com/ScriptReference/.

Unity. 2015d. Tutorials. Viitattu 10.12.2015 https://unity3d.com/learn/tutorials.

W3Techs. 2015. Usage of JavaScript for websites. Viitattu 16.12.2015 http://w3techs.com/technologies/details/cp-javascript/all/all.

Xenosmash Games. 2014. Scripting UI in Unity 4.6. Viitattu 9.12.2015 https://www.youtube.com/watch?v=D6ggcMdm9_4.

Liite 1. OnTrigger2D ohjelmakoodi

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class trigger : MonoBehaviour {

    public Text teksti;
    public float fadeSpeed = 5f;
    public bool entrance;
    public GameObject Tuto1;

    void Awake () {
        teksti = Tuto1.GetComponentInChildren<Text> ();
        teksti.color = Color.clear;
    }

    void Update () {
        ColorChange();
    }

    void OnTriggerEnter2D (Collider2D other) {

            entrance = true
    }

    void OnTriggerExit2D (Collider2D other) {
            entrance = false;
    }

    void ColorChange () {
        if (entrance)
        {
            teksti.color = Color.Lerp (teksti.color, Color.white, fadeSpeed * Time.deltaTime);
        }

        if (!entrance)
        {
            teksti.color = Color.Lerp (teksti.color, Color.clear, fadeSpeed * Time.deltaTime);
        }

    }
}
```

Liite 2. Objektin siirtymistoiminnon ohjelmakoodi

Warp.cs | SaveSystem.cs | Playercontroller.cs | trigger.cs

No selection

```
using UnityEngine;
using System.Collections;

public class Warp : MonoBehaviour {

    public Transform warpTarget;

    IEnumerator OnTriggerStay2D (Collider2D other){

        ScreenFader sf = GameObject.FindGameObjectWithTag ("Fader").GetComponent<ScreenFader> ();

        if (Input.GetKeyUp ("e")) {

            yield return StartCoroutine (sf.FadeToBlack ());

            other.gameObject.transform.position = warpTarget.position;

            yield return StartCoroutine (sf.FadeToClear ());

        }
    }

}
```